A problem $Y$ is **NP-hard** if for every problem $X \in \text{NP}$, $X \leq^P Y$.

A problem is **NP-complete** if it is both in $\text{NP}$ and $\text{NP-hard}$.

The goal of an optimization problem is to find the minimum (or maximum) value under some constraints.

$\text{OPT}(I)$ is the value of the optimal solution to an instance $I$ of an optimization problem.

We say an algorithm $\mathcal{A}$ for an optimization problem is a factor-$\alpha$ approximation if for all instances $I$ of the problem $\mathcal{A}$ outputs a solution that is at least as good as $\alpha \cdot \text{OPT}(I)$.

**Edge Cover-Up**

Let $G = (V, E)$ be a graph. A vertex covering of $G$ is a set $C \subseteq V$ such that for every edge $\{x, y\} \in E$, either $x \in C$ or $y \in C$ (a set of vertices such that every edge is incident to at least one vertex in the set). An independent set in $G$ is a set $S \subseteq V$ such that for any $u, v \in S$, $\{u, v\} \notin E$ (a set of vertices such that no edge connects two vertices in the set). Define the following languages:

**VERTEX-COVER:** $\{\langle G, k \rangle : G$ is a graph, $k \in \mathbb{N}^+; G$ contain a vertex covering of size $k\}$

**IND-SET:** $\{\langle G, k \rangle : G$ is a graph, $k \in \mathbb{N}^+; G$ contains an independent set of size $k\}$

Show that $\text{VERTEX-COVER} \leq^P \text{IND-SET}$.

We will define a map $f : \Sigma^* \to \Sigma^*$ such that $x \in \text{VERTEX-COVER} \iff f(x) \in \text{IND-SET}$.

```python
def f(x):
    if x is not an encoding $\langle G, k \rangle$ where $G$ is a graph and $k \in \mathbb{N}^+$,
        return $\varepsilon$ (Assuming for our encoding method, $\varepsilon \notin \text{IND-SET}$)
    let $n$ be the number of vertices in $G$
    return $\langle G, n - k \rangle$
```

- If $x \in \text{VERTEX-COVER}$ then $x$ is a valid encoding $\langle G, k \rangle$ where $G$ has $n$ vertices. In addition, there exists a vertex cover of size $k$. Let $S$ be the subset of vertices in the vertex cover and consider $V \setminus S$. Observe that $V \setminus S$ is an independent set. To see why, consider any pair of vertices $u, v \in V \setminus S$. Because neither $u$ nor $v$ are in $S$ but $S$ is a vertex cover, we know that $(u, v) \notin E$. In addition, $|V \setminus S| = n - k$, so there exists a independent set of size $n - k$. Hence $f(x) \in \text{IND-SET}$.

- If $f(x) \in \text{IND-SET}$, then $G$ must has an independent set of size $n - k$. Let $S$ be the subset of vertices in the independent set. Similar to the previous direction, $V \setminus S$ is a vertex cover. To show this, consider an arbitrary edge $\{u, v\} \in E$. Because $V \setminus S$ was an independent set, we know one of $u, v \notin S$. Equivalently, one of $u, v \in V \setminus S$. Thus, all edges are covered by some vertex in $V \setminus S$ so $G$ has a vertex cover of size $n - (n - k) = k$. So $x \in \text{VERTEX-COVER}$.

- $f$ is poly-time since we just need verify that the encoding is valid, count the number of vertices, and then compute $n - k$, which can all be done in poly-time.

**Cut and Dried**

We define the Max-Cut problem as follows:
Let $G = (V, E)$ be a graph. Given a coloring of the vertices with 2 colors, we say that an edge $e = \{u, v\}$ is a cut if $u$ and $v$ are colored differently. In the Max-Cut problem, the input is a graph $G$, and the output is a coloring of the vertices with 2 colors that maximizes the number of cut edges.

Consider the following approximation algorithm for the Max-Cut problem:

```
function MaxCutApprox(G = (V, E))
    color all vertices red  # allowable colors are red and something else (say, blue)
    while there exists $v \in V$ s.t. changing $v$'s color increases the number of cut edges do
        change the color of $v$
    end while
    return coloring
end function
```

(a) Show that this algorithm is poly-time.

(b) Prove that this algorithm is a $\frac{1}{2}$-approximation for Max-Cut.

(c) Show that this algorithm is not a $(\frac{1}{2} + \varepsilon)$-approximation algorithm for Max-Cut for any $\varepsilon > 0$.

See Chapter 13 for a solution to (a) and (b).

Towards (c), let $k$ be arbitrary. Informally, draw $k$ copies of $C_6$ arranged in a line, and then connect all middle vertices in a path (an example when $k = 3$ has been drawn below). A poor algorithm trace colors all the vertices on the path red and the rest of the vertices blue, achieving $4k$ cut edges; it is easy to verify that it is possible for the algorithm to do this, and once it reaches this point it will terminate. Observe that there are $8k - 1$ edges in total, and since the graph is bipartite the optimal solution will cut all the edges. So the approximation ratio for this algorithm cannot exceed $\frac{4k}{8k - 1}$; taking $k$ sufficiently large shows that this is not a $(\frac{1}{2} + \varepsilon)$-approximation for any constant $\varepsilon > 0$.

```
Gotta Catch a Lot of 'Em
```

Consider a set of Pokémon and a set of $m$ trainers each having a subset of these Pokémon. Given an integer $k$, the problem is to pick $k$ trainers in a way that maximizes the number of distinct Pokémon owned among them. This problem will show that there exists a poly-time $(1 - 1/e)$-approximation by considering the following greedy algorithm:

```
function PokémonApprox((S_1, S_2, ..., S_m), k)
    T ← ∅  # keeps track of trainers we have already picked
    U ← ∅  # keeps track of which Pokémon we have already picked
    for $1 \leq i \leq k$ do
        $j ← \arg \max_j |S_j - U|$  # pick the trainer $j$ with the most new Pokémon
        $T ← T ∪ \{j\}$
        $U ← U ∪ S_j$
    end for
    return $T$
end function
```
(a) Prove that the algorithm runs in poly-time.

(b) Let $T^*$ denote the optimum solution, and let $U^* = \bigcup_{j \in T^*} S_j$. Further, define $U_i$ to be the set $U$ in the algorithm after the $i$-th iteration of the loop. Prove that $|U^*| - |U_i| \leq (1 - \frac{1}{e})^i |U^*|.$

(c) Using the inequality $1 + x \leq e^x$, deduce that this algorithm is a $(1 - \frac{1}{e})$-approximation.

See Chapter 13 of the notes.

**(Extra) Looping Around**

Show that the HALTS is NP-hard.

First observe that 3-SAT is decidable. Given an input, we can simply try all $2^n$ possibilities for the variable assignment, then check all the clauses. We’ll present the reduction in a slightly different way: we’ll give a decider $S$ for 3-SAT that returns precisely the result of a black-box call to a decider $H$ for HALTS (make sure you understand why this notion is equivalent). So consider the following machine:

```python
def S(x):
    def HELP(y):
        brute force to check if $x$ is satisfiable
        if $x$ is satisfiable, then halt
        if $x$ is not satisfiable, then loop
        return $H(<HELP, "garbage string">)$
```

Observe that the input to $H$ is indeed polynomial in the size of the 3-SAT input $x$, as it only requires $x$ to be hard coded once and the remainder of it is constant in size. Further, if it halts, then the expression is satisfiable, and vice versa. Hence, if HALTS can be decided in polynomial time, then 3-SAT can also be decided in polynomial time. Since 3-SAT is NP-complete, it follows that HALTS is NP-hard.

**(Bonus) Hard Cut**

On the previous page, we defined Max-Cut as an optimization problem. We can also define the decision version MAX-CUT as follows:

**MAX-CUT**: $\{\langle G, k \rangle : G$’s vertices may be colored with two colors in a way that cuts at least $k$ edges$\}.$

Prove that MAX-CUT is NP-hard. This is slightly difficult; try reducing from IND-SET.
We'll present a gadget reduction. Given an instance $\langle G = (V,E), k \rangle$ of IND-SET, we'll output an instance $\langle G', k' \rangle$ of MAX-CUT such that $\langle G, k \rangle \in$ IND-SET $\iff \langle G', k' \rangle \in$ MAX-CUT. $G'$ is defined as follows: let $s$ be a new vertex, and for each $v \in V$ create a node labeled $v$ in $G'$, and draw the edge $\{v, s\}$. For each edge $\{u, v\} \in E$, we create the following gadget and insert it in $G'$:

$$
\begin{align*}
&u \quad (uv)_u \\
&s \\
&v \quad (uv)_v \\
&w \\
\end{align*}
$$

For example, we transform the following graph as shown ($G$ on the left, $G'$ on the right).

We then output $\langle G', k + 4|E| \rangle$. Before proving correctness, first observe that this is constructible in poly-time, since we add one gadget per edge. Also, note that for a given gadget, if neither $u$ nor $v$ are colored the same as $s$, then at most three of the five edges are cut. On the other hand, if at least one of them are colored the same as $s$, then it is possible to color the intermediary vertices such that four of the five edges are cut. As a corollary, observe that for each gadget, at most four edges can be cut.

($\Rightarrow$) Now, suppose that there exists an independent set $S$ of size $k$ in $G$. Color all the vertices corresponding to those in $S$ red, and the vertices corresponding to those in $V - S$ blue. Also, color $s$ blue. Since $S$ is an independent set, for every gadget, at least one of its vertices is blue. Then we may color the intermediary vertices such that four of the five edges in the gadget are cut. The only other edges in $G'$ are those connecting vertices in $V$ to $s$; since $S$ is of size $k$, there are $k$ such cut edges. This coloring therefore achieves $k + 4|E|$ cut edges, and so $\langle G', k + 4|E| \rangle \in$ MAX-CUT, as desired.

($\Leftarrow$) In the reverse direction, suppose that $\langle G', k + 4|E| \rangle \in$ MAX-CUT. Since each gadget can only have four of its edges cut, there are at least $k$ edges cut among the non-gadget edges. Set $S = \{v : \{v, s\}$ is cut$\}$, and write $|S| = k + \ell$ for some $\ell \geq 0$. Then there are at most $\ell$ edges such that both endpoints are in $S$; if there were more, then we would not be able to achieve $k + 4|E|$ cut edges. For each such edge, delete one endpoint arbitrarily from $S$. The remaining set is independent and has at least $k + \ell - \ell = k$ vertices, so we're done.